From Sequence Diagrams to Behaviour Models

Sebastian Uchitel, Jeff Magee and Jeff Kramer
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK

1 INTRODUCTION

Sequence Diagrams

The software engineering community has long understood the importance of requirements elicitation. Stakeholder involvement in the elicitation process and tools to help build a common ground between stakeholders and developers is essential in order to obtain a good requirements definition. Scenarios have become increasingly popular as a means of articulating stakeholder requirements. Scenarios describe how system components (in the broadest sense) and users interact in order to provide system level functionality. Each scenario is a partial story which, when combined with all other scenarios, should conform to provide a complete system description. Thus stakeholders may develop descriptions independently, contributing their own view of the system to those of other stakeholders.

The Unified Modelling Language has a notation for scenarios called *Sequence Diagrams* [1]. These diagrams, together with their counterpart from the telecommunication industry *Message Sequence Charts* [2], have become widely accepted notations for scenario-based specification. Although sequence diagrams facilitate the requirement elicitation process, they have not been exploited to their full extent for requirement analysis and for transitioning into the design phase. This is due fundamentally to the lack of tool support and the lack of agreement on the exact meaning of this graphical notation.

Behaviour Models

Modern software systems tend to be of a highly complex and concurrent nature, and often have strict correctness requirements. Pre-deployment and pre-development reasoning about system behaviour is crucial in application areas such as industry, avionics, health care, and defence where the cost of failure is extremely high. However, *the wide acceptance of Java with its in-built concurrency constructs means that concurrent programming is no longer restricted to the minority of programmers involved in operating systems and embedded real-time applications* [3]. Thus, there is further need to provide accessible technology for understanding the subtle properties of the concurrent system behaviour.

The main principle behind analysis of concurrent system behaviour is the construction of models. These models are simplified representations that focus on the interactions between components working concurrently. They are usually called behaviour models as they describe how components behave with respect to other components. If rigorous analysis is to be performed, behaviour models must be formally defined. In other words, they must be based on mathematical modelling techniques and have well-understood properties. A behaviour model can be used as a precise specification of intended behaviour, as a prototype for exploring the system behaviour and also to allow for automated checking of model compliance to properties (*model checking*). Numerous tools that allow model checking and animation of behaviour models exist (e.g. [3-6]).

Sequence Diagrams to Behaviour Models

Scenarios view systems as collections of independent, concurrent components and show how they interact in order to provide system level functionality. This view coincides with that of behaviour models for concurrent systems. Nevertheless, although it is clear that there is an overlap between scenario specifications and behaviour models, the precise relation between these two is usually unclear. Our general goal is to try to clarify the relationship.

There seems to be an interesting balance between the two in terms of potentials and shortcomings. Scenario specifications are still maturing with respect to the definition of

rigorous semantics and analysis tools, however they already have wide acceptance in industry. Behaviour models have not yet had a major impact on practitioners, nevertheless boast an important mathematical foundation and efficient tools for behaviour analysis. We believe that understanding the relation between scenarios and behaviour models can help define techniques and tools that leverage both areas.

2 RELATED WORK

There has been much work on scenarios. Publications deal with subjects that range from formal semantics to informal development methodologies. In general, they all agree on what is a scenarios (in its most simplest sense) and how it should be interpreted. However, from then on there are more differences than similarities. Scenarios specification languages have been enriched by a variety of constructs that include specification of alternatives, loops, timers, message loss, component creation and destruction, compositional constructs and data [7, 8]. Besides, as scenarios are enriched with more complex features, their semantics becomes unclear. Even in the presence of very basic features, important differences in interpretation occur. This is especially true when considering a set of scenarios and how they relate.

In the approach adopted by the International Telecommunication Union (ITU) [2] and others [9-11], focus is on providing scenario-based specifications with a means for managing complexity. *Basic Message Sequence Charts* (bMSCs) are used to specify simple sequences of behaviour whilst *High-level Message Sequence Charts* (hMSCs) are used to indicate their possible orderings. hMSCs allow stakeholders to reuse scenarios within a specification and to introduce sequences, loops, and disjunctions of bMSCs [2]. The advantage of the hMSC approach is that it allows stakeholders to break up a scenario specification into manageable parts in a simple, intuitive, and operational way, and to show how these different parts relate. A diffent approach is presented in [12-14], where focus is on identifying, throughout the set of scenarios, those states that are considered to refer to the same component state. For example, Whittle and Schumann [12] use the Object Constraint Language (OCL) to express pre- and post-conditions for messages. These are traversed with bMSCs to produce a valuation of global state variables in bMSC states. These valuations are used to identify equivalent states. Another example is the statechart synthesis algorithm in SCED [14]. This approach employs the domain-specific assumption that the capability of outputting a specific message uniquely identifies the state of a component.

In terms of semantics of scenario-based specifications, there are several approaches: In some approaches scenario notations are used with no well-defined semantics as for example in some UML-based development methodologies such as [1, 15, 16]. These approaches allow documentation and communication of requirements but are hard to use for rigorous analysis as their meaning is unclear. In other cases, algorithms are provided for translating scenarios into other notations [12, 14, 17]. These approaches provide more insight to the meaning of scenarios if the target notation has a well-defined semantics. However, this procedure, called synthesis, is rather operational and can (and usually does) hide in the synthesis algorithm many subtle aspects of the scenario semantics. We believe that a better approach is to define a declarative semantics for the scenario specification language and to construct a sound synthesis algorithm with respect to the semantics. Among the approaches using synthesis, there are several approaches that generate statechart models [12-14, 17]. Authors argue that statecharts provide a more structured, and therefore more understandable, view of component behaviour. However the drawback is that statechart semantics (based on micro and macro-steps) is rather complex and availability of automated analysis tools that support the formalism is limited. An interesting aspect of synthesis is that it offers the possibility of using additional information in the form of alternative specifications [12, 13] or domain-specific assumptions [14] to produce behaviour models that integrate different information sources. Finally, several formal semantics for scenario languages have been proposed. In [11] the semantics complies with a *delayed choice* policy. Meaning that a component, when choosing between two different possible scenarios, will postpone the decision if both scenarios have common initial events. Although delayed choice is a reasonable assumption in many cases, there are some

situations where non-determinism is desirable. The formal semantic definition is given in terms of process algebra using non-standard operators to model delayed choice. Other formalisations exist, both using delayed choice (e.g. [18, 19]) and not (e.g. [20]).

In terms of automated analysis of scenario-based specifications there has not been so much work. Some approaches focus on detecting some consistency criteria (e.g. [9]) which is done syntactically. Other approaches such as in [21] focus on checking specific properties such as process divergence and non-local choice. However, in these approaches, there is no construction of a model that can then be checked for consistency or analysed with respect to ad-hoc system properties which might be proposed by designers or stakeholders.

3 WORK IN PROGRESS

One of our objectives is to facilitate the development of behaviour models in conjunction with scenarios. Being scenarios complementary to such models, in addition to providing an alternative view, we believe that there is benefit to be gained by experimenting with and replaying analysis results from behaviour models in order to help correct, elaborate and refine scenario-based specifications.

Our initial focus has been on existing approaches to scenario specifications and on understanding the rationale behind the many assumptions and uses they have. It became clear that our approach should try to integrate existing ones by providing a core language on which other languages could be built on. In [22] our aim has been to provide a workbench for supporting various approaches to scenario-based specifications, behaviour synthesis and analysis. We have defined a formal semantics for a scenario language that integrates approaches based on high-level message sequence charts and on identifying component states. However, instead of assuming specific criteria for identifying component states, we provide a simple mechanism for making this information explicit within a sequence diagram using state labels [2]. In this way we aim to provide a workbench for approaches such as [12-14] that allows for explicit additional information (usually in some other formalism such as OCL) and/or domain-specific or other assumptions within an scenario-based specification. Furthermore, we show how many of these assumptions can be automatically translated into state labels. The semantics is given in terms of Labelled Transition Systems (LTS) and parallel composition [23], which are fully understood and widely accepted mathematical constructs for modelling concurrent systems. In addition, we have developed an algorithm for the automatic synthesis of system behaviour models. We have integrated our synthesis process to an existing model checking tool to support system requirements validation. This is done by first translating the specification into a Finite Sequential Processes (FSP) specification [3], which can then be analysed using the Labelled Transition System Analyser [3] by model checking for deadlock, safety and liveness properties and by model animation [24]. In [25] we have shown the soundness of our synthesis algorithm with respect to the language semantics.

4 FUTURE WORK

Scenarios have proved to be a good tool for bridging the gap between stakeholders and developers. However, up to now, this is mainly a one-way bridge in which developers gain more insight of stakeholders' domain knowledge. Future work will be focused on building a bridge in the other direction, i.e. building mechanisms to provide feedback of the developer's world to stakeholders. Preliminary work in this direction is promising. We are automating the construction of alternative system views from synthesised LTS models. Interestingly, taking advantage of the semantic overlap between high-level sequence diagrams and state labels, one can generate many different views. State labels identify component states across scenarios, while high-level sequence diagrams provide information about all components by relating scenarios. Moving information from one representation to the other allows for a large number of possible views that vary from long scenarios that start at the system's initial state to short scenarios that optimise reuse. These views can allow stakeholders to gain more insight into their own scenario specifications or be used by designers to show the impact of their changes to behavioural models in a language that stakeholders manage.

The use of state labels and high-level sequence diagrams to add information on the branching structure of components suggests that it may be useful to develop some methodological guidelines for incrementally building a complete behaviour model from an initially scarce set of simple scenarios. We are looking into this aspect and hope to develop some tools and techniques to facilitate this process.

Finally there is an important extension that we shall be looking at which is the inclusion of time into scenarios. This may allow us to develop a whole new set of tools and techniques for real-time systems.

5 CONCLUSION

We believe that scenario-based specifications and behaviour models can complement each other, providing alternative views of concurrent systems, models for experimentation and analysis in order to help correct, elaborate and communicate system requirements. To enable this, it is important to understand the relationship between scenarios and behaviour models and to define techniques and tools that can leverage the advantages of both areas.

We have defined a formal semantics for a sequence diagram-based language that serves as a workbench for supporting various other approaches to scenario-based specification, behaviour synthesis and analysis. We have also developed a synthesis algorithm integrated with the LTSA model-checking tool that permits behaviour model analysis. We are now currently working on generating feedback from behaviour models in form of scenario-based specifications.

REFERENCES

1. Booch, G., J. Rumbaugh, and I. Jacobson, *The Unified Modelling Language User Guide*, ed. Addison-Wesley. 1998.
2. ITU, *ITU-T Recommendation Z.120. Message Sequence Charts (MSC'96)*. 1996, ITU Telecommunication Standardisation Sector: Geneva.
3. Magee, J. and J. Kramer, *Concurrency: State Models and Java Programs*. 1999, New York: John Wiley & Sons Ltd.
4. Kramer, J. and J.C. Cheung. *Compositional reachability analysis of finite-state distributed systems with user specified constraints*. *SIGSOFT*. 1995. Washington D.C.
5. Burch, J.R., *et al.*, *Symbolic model checking: 10^20 and beyond.* Information and Computation, 1992(98): p. 142-170.
6. Holzmann, G.J. and D. Peled, *The state of Spin*. Prentice Hall Software Series, ed. Prentice-Hall. 1991.
7. Mauw, S. *The Formalization of Message Sequence Charts*. *1st Workshop of the SDL Forum Society on SDL and MSC*. 1998. Berlin, Germany.
8. Haugen, O., *MSC-2000 Interaction for the new Millenium*. 2000, SDL Forum MSC2000.
9. Alur, R., G.J. Holzmann, and D. Peled. *An Analyser for Message Sequence Charts*. *Second International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*. 1996. Passau, Germany.
10. Rudolph, E., P. Graubmann, and J. Grabowski. *Tutorial on Message Sequence Charts '96*. *FORTE/PSTV*. 1996. Kaiserslautern, Germany.
11. Cobens, J.M.H., *et al.*, *Formal Semantics of Message Sequence Charts*. 1998, Eindenhoven University of Technology: Eindhoven, The Netherlands.
12. Whittle, J. and J. Schumann. *Generating Statechart Designs from Scenarios*. in *22nd International Conference on Software Engineering (ICSE'00)*. 2000. Limerick, Ireland: ACM Press.
13. Somé, S., R. Dssouli, and J. Vaucher. *From Scenarios to Timed Automata: Building Specifications from User Requirements*. *Asia Pacific Software Engineering Conference*. 1995.
14. Systä, T., *Static and Dynamic Reverse Engineering Techniques for Java Software Systems*, in *Dept. of Computer and Information Sciences*. 2000, University of Tampere.

15. Texel, P.P. and C.B. Williams, *Use Cases Combined with Booch, OMT, and UML*. 1997: Prentice-Hall.
16. Quatrani, T., *Visual modelling with Rational Rose 2000 and UML*. 1998, Reading, Mass.: Addison Wesley.
17. Broy, M., *et al. From MSCs to Statecharts*. *Distributed and Parallel Embedded Systems*. 1999: Kluwer Academic Publishers.
18. Heymer, S. *A Non-Interleaving Semantics for MSC*. *1st Workshop of the SDL Forum Society on SDL and MSC*. 1998. Berlin, Germany.
19. Katoen, J.-P. and L. Lambert. *Pomesets for Message Sequence Charts*. in *1st Workshop of the SDL Forum Society on SDL and MSC*. 1998. Berlin, Germany.
20. Alur, R., K. Etessami, and M. Yannakakis. *Inference of Message Sequence Charts*. *22nd International Conference on Software Engineering (ICSE'00)*. 2000. Limerick, Ireland.
21. Ben-Abdhallah, H. and S. Leue. *Syntactic Detection of Process Divergence and Non-Local Choice in Message Sequence Charts*. *Third International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*. 1997: Springer-Verlag.
22. Uchitel, S. and J. Kramer. *A Workbench for Synthesising Behaviour Models from Scenarios*. *ICSE 2001*. 2001. Toronto, Canada.
23. Milner, R., *Communication and Concurrency*. International Series in Computer Science. 1989: Prentice-Hall.
24. Magee, J., *et al. Graphical Animation of Behaviour Models*. *22nd International Conference on Software Engineering (ICSE'00)*. 2000. Limerick, Ireland.
25. Uchitel, S. and J. Kramer, *A Sound Algorithm for Synthesis of Behaviour Models from Scenarios*. 2001, Department of Computing, Imperial College. London, UK.